# QUBO Implementation of Traveling Salesman Problem Using Variational Quantum Eigensolvers

Akshay Kushwaha, Osewulke Igue, Nabin Pandey

May 2025

**Abstract**

This document details the implementation of a Quadratic Unconstrained Binary Optimization (QUBO) formulation for the Traveling Salesman Problem (TSP) using Python and SymPy. We present a complete mathematical formulation, constraint derivation, and objective function construction for solving TSP on quantum computers. The implementation includes distance matrix processing, constraint coefficient calculation, and symbolic polynomial generation for quantum optimization. Our approach demonstrates the fundamental steps required to transform a classical optimization problem into a form suitable for quantum algorithms like VQE.

# 1 Introduction

## 1.1 Problem Overview

The Traveling Salesman Problem (TSP) is an NP-hard optimization problem where given a list of cities and distances between each pair, we seek the shortest possible route that visits each city exactly once and returns to the origin city.

### 1.2 QUBO Formulation

The QUBO formulation transforms the TSP into a binary optimization problem suitable for quantum computing. Our implementation:

- Encodes city visits as binary variables
- Formulates distance minimization as a quadratic objective
- Adds penalty terms for constraint satisfaction

## 2 Implementation

### 2.1 Distance Matrix Setup

```
# Adjacency matrix representing distances between cities
adjList = np.array([
    [0, 10, 50, 45],
    [10, 0, 25, 25],
    [50, 25, 0, 40],
    [45, 25, 40, 0]
])
```

Listing 1: Distance matrix initialization

### 2.2 Constraint Calculation

The constraint coefficients (A, B, C) are derived from edge distance bounds:

```
# Calculate upper and lower bounds for constraints
edges = []
n = adjList.shape[0]
for i in range(n):
    for j in range(i + 1, n):
        edges.append((i, j, adjList[i, j]))

sorted_edges = sorted(edges, key=lambda x: x[2])
sorted_edges_desc = sorted(edges, key=lambda x: x[2], reverse
    =True)

LB_edges = sorted_edges[:4]
LB = sum(edge[2] for edge in LB_edges)

```

```
UB_edges = sorted_edges_desc[:4]
UB = sum(edge[2] for edge in UB_edges)

A = B = C = UB - LB + 1  # Constraint coefficients
```

Listing 2: Constraint coefficient calculation

## 2.3 Objective Function

The complete QUBO objective function with constraints:

```
def objective_function(x, y, dist):
    return (
        # Distance term
        sum(dist[i][j] * x[i,j] for i in range(len(dist))
                            for j in range(len(dist)) if i != j
    ) +

        # City visit constraints (each city visited exactly
    once)
        A * (x[1,0] + x[2,0] + x[3,0] - 1)**2 +
        A * (x[0,1] + x[2,1] + x[3,1] - 1)**2 +
        A * (x[0,2] + x[1,2] + x[3,2] - 1)**2 +
        A * (x[0,3] + x[1,3] + x[2,3] - 1)**2 +

        # Time slot constraints (one city per time)
        B * (x[0,1] + x[0,2] + x[0,3] - 1)**2 +
        B * (x[1,0] + x[1,2] + x[1,3] - 1)**2 +
        B * (x[2,0] + x[2,1] + x[2,3] - 1)**2 +
        B * (x[3,0] + x[3,1] + x[3,2] - 1)**2 +

        # Additional constraints
        C * x[1,1]**2 + C * x[2,2]**2 + C * x[3,3]**2 +
        C * (x[1,1] + x[1,2] + x[2,1] + x[2,2] + y[0] - 1)**2
    +
        C * (x[1,1] + x[1,3] + x[3,1] + x[3,3] + y[1] - 1)**2
    +
        C * (x[2,2] + x[2,3] + x[3,2] + x[3,3] + y[2] - 1)**2
    +
        C * (x[1,1] + x[1,2] + x[1,3] + x[2,1] + x[2,2] + x
    [2,3] +
             x[3,1] + x[3,2] + x[3,3] + y[3] + y[4] - 2)**2
    )
```

Listing 3: QUBO objective function

# 3 Mathematical Formulation

The symbolic representation of the QUBO problem:

$$\begin{aligned} H = & \sum_{i,j} d_{ij} x_{ij} \quad \text{(Distance term)} \\ & + A \sum_{t} \left( 1 - \sum_{i} x_{i,t} \right)^2 \quad \text{(City visit constraints)} \\ & + B \sum_{i} \left( 1 - \sum_{t} x_{i,t} \right)^2 \quad \text{(Time slot constraints)} \\ & + C \sum \text{(Additional constraints)} \end{aligned} \tag{1}$$

The complete polynomial expansion is generated using SymPy:

```
X = MatrixSymbol('X', n, n)
Y = MatrixSymbol('Y', 1, 5)
sym_objective_function = poly(objective_function(X, Y,
    adjList))
```

Listing 4: Symbolic polynomial generation

# 4 Results and Discussion

## 4.1 Implementation Notes

- Successfully formulated TSP as QUBO problem
- Derived constraint coefficients from distance matrix bounds
- Generated complete symbolic polynomial for quantum optimization

## 4.2 Performance Considerations

- Constraint coefficients (A, B, C) ensure valid solutions dominate
- Quadratic terms properly encode the optimization landscape
- Polynomial grows quickly with problem size (n cities $\rightarrow$ $n^2$ variables)

# 5 Conclusion

This implementation demonstrates:

- Complete QUBO formulation of TSP
- Proper constraint handling through penalty terms
- Preparation for quantum optimization via VQE

Future work includes:

- Quantum circuit implementation
- Optimization with different ansatzes
- Noise resilience analysis